1993

Broadcast Methods in the Multiswitch Point-to-Point Network

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

Victor Norman

Report Number:
93-030

BROADCAST METHODS IN THE MULTISWITCH POINT-TO-POINT NETWORK

Douglas E. Comer
Victor T. Norman

CSD-TR-93-030
May 1993

Broadcast Methods in the Multiswitch Point-to-Point Network

Douglas Comer
Victor Norman

Purdue University
West Lafayette, IN 47907

CSD-TR-93-030
May 21, 1993

The Multiswitch project at Purdue University is investigating broadcast techniques for use in point-to-point networks. This paper analyzes four techniques for delivery of a broadcast message to all processors in a network. The current Multiswitch prototype network uses one of these techniques, called Flooding with Duplicate Discard (FDD). FDD is attractive because it provides high reliability of delivery of the broadcast message to all processors while requiring a relatively low number of packets. In FDD, the number of packets required to complete a broadcast is a linear function of the number of communication links and processors in the network, independent of the source of the broadcast. Each FDD broadcast traverses a *source tree*, a tree rooted at the source of each broadcast containing paths with minimum delay to all processors in the network. The source tree for a broadcast is *not* configured prior to the broadcast. An FDD broadcast message carries a unique identifier that must be held in a cache on each processor, but, for most networks, the identifier need not be held for long periods of time. An extension to FDD allows for efficient implementation of another broadcasting technique, source-based forwarding.

1. Introduction

The Multiswitch project is investigating and building a network architecture that uses point-to-point links and store-and-forward packet switching. A Multiswitch prototype network consists of multiple, geographically-distributed packet switches interconnected by point-to-point high-speed links. As figure 1 shows, a packet switch, in the current prototype, consists of six processors connected in a *wheel* configuration by high-speed communication links. Five of the six processors connect to local-area networks. Three of the processors have fiber-optic connections. The processors in a Multiswitch network require complete network topology information, as well as *link-state* information, the current operating characteristics for each communication link in the network. Each processor monitors its own outgoing links and periodically broadcasts the link-state information to all other processors in the network. As the number of processors in the network grows, the load that each broadcast imposes upon the network increases. We have investigated various, well-known broadcasting techniques in an effort to minimize the network resources required by each broadcast.

For our purposes, broadcast methods must meet two basic criteria. First, a method should ensure with a high probability that the message reaches all processors in the network. Second, a broadcast method should ensure that unnecessary duplicate copies of the message are either not generated or quickly suppressed. [MRR80] identified these two criteria as *high reliability* and *high efficiency*. We use these two criteria to evaluate broadcast methods in common use.

This paper discusses the advantages and disadvantages of the various, well-known broadcast methods for point-to-point networks. It then defines FDD and discusses some interesting properties of FDD. It also addresses the problems with FDD mentioned by Dalal and Metcalfe [DM78] and presents solutions to those problems. The appendix documents an extension of FDD that allows efficient implementation of source-based forwarding.

The paper proceeds as follows. Section 2 defines terminology used throughout the paper. Section 3 details the criteria used for evaluation of the various broadcast methods, while section 4 outlines some assumptions to make when evaluating the methods. Section 5 analyzes three broadcasting techniques in common use. Familiarity with these techniques is useful for later comparison to the FDD method. Section 6 defines FDD, the method chosen for use in the Multiswitch prototype network, and points out the interesting properties of FDD. The appendix contains an extension of FDD that can be used to efficiently implement source-based forwarding.

2. Terminology

A *processor* is a computer that forwards packets from one physical communication medium to another. A *link* is the communication medium that connects two processors. A processor that sends a broadcast is called the *source* of the broadcast.

A *broadcast* refers to the single action of a source processor when it needs to send a message to all other processors in the network. The message a processor broadcasts is a *broadcast message*. A *packet* is an individual copy of the broadcast message that traverses a link in the network.

The set of *outgoing links* for a processor and a particular broadcast packet received at the processor is all of the processor's links except the link on which the packet arrived. The term *flooding* refers to the action a processor takes when it receives a broadcast packet, creates multiple copies of the packet, and sends the copies out over its outgoing links. *Forwarding*, on the other hand, means that a processor makes copies of a packet and sends them out on a select subset of its outgoing links.

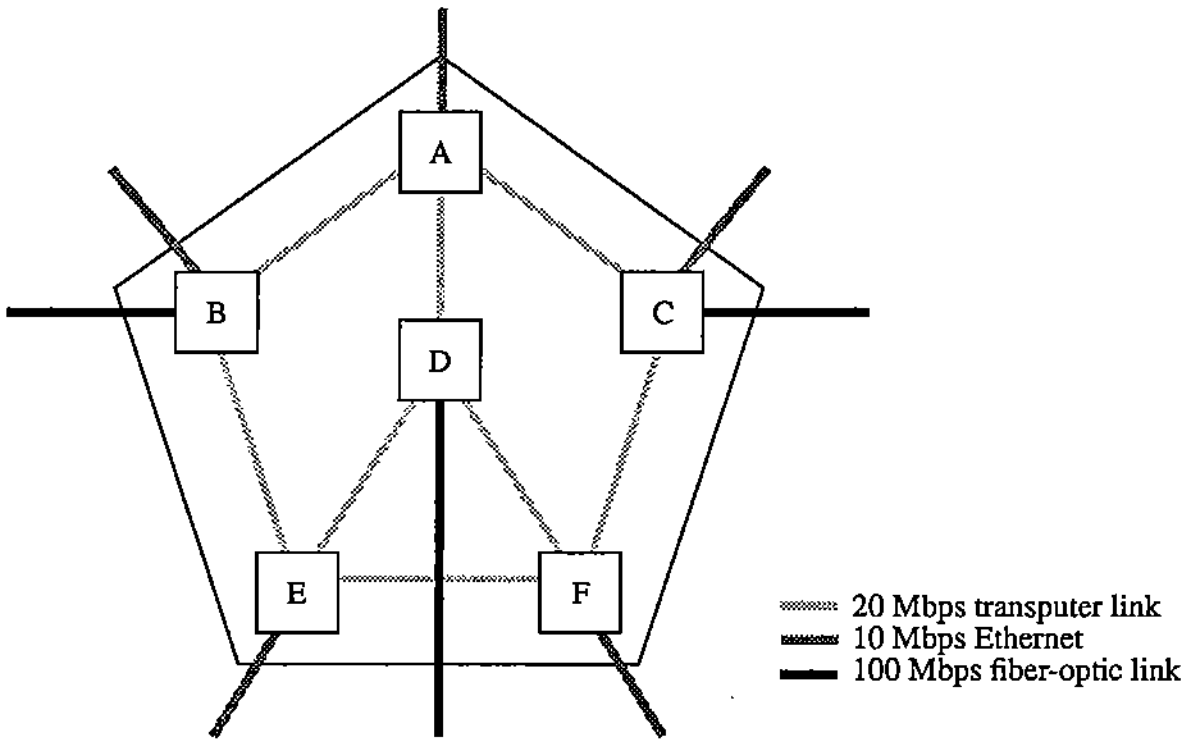

Figure 1: Packet switch architecture called a *wheel* configuration

Link delay is the time required for a processor to process a packet, place the packet in a queue of outgoing packets destined for a physical link, and transmit the packet over the physical link. Link delay also includes the time the packet spends waiting in a queue after being transmitted and before being processed at the neighbor processor. In a Multiswitch network, the packet transmission time and packet processing times remain fairly constant and are small when compared to the maximum queueing time.

An undirected graph models a point-to-point network, where the nodes in the graph represent processors and weighted edges represent bidirectional communication links. On a graph, a *shortest path* between two nodes is the path with minimum weight between the two nodes. A *source tree* for a node s is a tree rooted at s that reaches all nodes in the graph with minimum path weight [Raj92]. [Moy91] calls a source tree a *Shortest-Path First tree*. In a network, the source tree for a processor s is a tree rooted at s that reaches all processors in the network with minimum delay.

3. Evaluation Criteria

We evaluate each broadcast method according to the following criteria, ordered from most important to least important.

- **Number of Packets Required:** Under controlled circumstances, each method requires a fixed number of packets to successfully deliver the broadcast message to all processors. An efficient method generates few unnecessary duplicate copies.

- **Reliability of Delivery:** In some schemes, a dropped packet may cause processors in the network to never receive the broadcast. More reliable methods take steps to reduce the probability of this

occurrence.

- **Computation Required:** Each method requires some computation to handle and forward a broadcast. Methods that require little computation decrease the load a broadcast places on the network.

- **Memory Required:** Some methods require many resources at each processor to hold data structures. A method that requires little or no memory per processor leaves more memory free for packet storage.

- **Packet Modification:** Some methods require specific fields to be present in each broadcast packet header, while others require that each processor modify fields during broadcast forwarding. Methods that do not require modifications to special fields increase the efficiency of broadcast delivery and reduce the possibility of errors.

4. Assumptions

When evaluating a broadcast method, we make certain assumptions about the network in which the broadcast method operates. The assumptions reflect a Multiswitch network as described in the introduction.

- **Best-Effort Delivery:** The underlying protocols provide best-effort delivery of packets. Best-effort delivery [Nar88] means that a processor exerts its best effort to deliver each packet correctly, but does not guarantee that packets will not be lost, duplicated, corrupted, or reordered. Errors in a Multiswitch network occur primarily from packet loss. Packet loss occurs due to buffer overruns in a congested processor.

- **Network Partitions:** The network does not become partitioned. (If a network becomes partitioned, *no* method can deliver a broadcast packet to all processors.)

- **Delay Used as Metric:** A network can use link delay, link bandwidth, link error rate, or some other link characteristic as the metric by which the network computes routes. A Multiswitch network uses link delay as the metric.

- **Presence of Routing Information:** For use in routing, each processor in the network maintains a copy of the complete topology of the network, as well as current link-status information for all links in the network. Certain broadcast methods, such as reverse path forwarding, work correctly only if this routing information is current.

5. Three Broadcasting Techniques in Common Use

This section describes and analyzes three broadcasting techniques in common use. The analysis presented here is provided mainly for later comparison to the Flooding with Duplicate Discard method described in Section 6.

5.1. Source-Based Forwarding

Source-based forwarding (SBF), described by Dalal and Metcalfe [DM78], performs a broadcast with minimum delay using the optimal number of packets. The optimal number of packets in a network containing n processors is n - 1. SBF requires forwarding information at each processor for each source in the network. The forwarding information for a broadcast from a particular

source consists of a set of links on which to forward the message. For each source, the forwarding information on all the processors creates a source tree that reaches all processors in the network with the minimum delay. A processor does not modify packets during delivery.

When a packet reaches a processor, the processor maps the source address of the packet to a subset of the outgoing links. The set defines the next set of links in the tree from the source of the packet. The processor copies the packet and forwards it on those links. Computation at each processor during delivery is low.

SBF has four major disadvantages. First, the method requires that a link set be present at each processor for each broadcast source. Each processor may compute its own link sets, or a central processor may compute the sets and distribute them over the network to each processor. The algorithm for computing the link sets, usually a variation of Dijkstra's algorithm [Dij59], requires $O(n^2)$ steps. Additionally, if broadcasts are to remain optimal at all times, processors must install new link sets each time there is a change in the network.

The second major disadvantage of SBF is the memory required to store link sets on each processor. The memory required grows proportional to the number of processors in the network because each processor must keep a link set for each source processor in the network. In some systems, this memory requirement may be unacceptable.

Third, source-based forwarding is susceptible to failure in the following situation. When routing information changes while the network is delivering a broadcast, new outgoing link sets in some processors may incorrectly cause a packet to be dropped, or may cause a packet to enter a *routing loop* where neighboring processors repeatedly send the packet back and forth to one another [DM78]. Therefore, source-based forwarding should be augmented by some other mechanism, such as a hopcount limit (see section 5.2), to stop packets that enter routing loops. The extra mechanism requires additional computation each time a processor forwards a packet.

Fourth, source-based forwarding does not provide high reliability of delivery in a best-effort delivery network. If a processor p drops a packet, the subtree of the source tree with root at processor p does not receive the broadcast. Figure 2 illustrates that a single dropped packet can cause most processors in the network to miss a broadcast.

5.2. Flooding With A Hopcount Limit

Flooding a network with a message means that each processor forwards a copy of the message over all links except the link from which the processor received the message. If not controlled, flooding may cause the number of packets generated in the network to grow exponentially as a function of the number of hops each packet takes. In the Flooding with a Hopcount Limit (FHL) broadcast method, the network suppresses the number of packets generated by limiting the number of links (hops) that each packet may traverse. The packet header includes a hopcount field. Each time a processor forwards the packet over a link, it decrements the hopcount. When the hopcount field reaches zero, the processor discards the packet.

If the initial hopcount limit is large, FHL provides a high degree of reliability of delivery, essentially guaranteeing that all processors will receive the broadcast message. On the other hand, some processors will not receive the broadcast message if the initial hopcount limit is less the number of hops to the processor farthest from the source of the broadcast.

Another advantage of FHL is that it requires no data structures at the processors. Each packet contains all the information required for each processor to make its forwarding decision. The amount of computation at each processor is also low because FHL requires no table lookups.

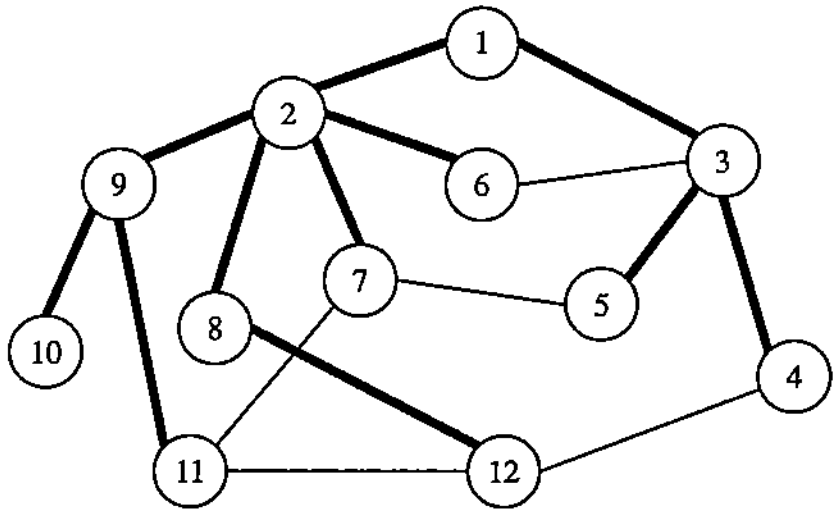

Figure 2: A network, with a source tree rooted at processor 1 superimposed on it. Bold lines represent links in the source tree; thin lines represent other links in the network. If processor 1 issues a broadcast packet and processor 2 drops the packet, processors 6 through 12 do not receive the broadcast. Only processors 3, 4, and 5 do receive the broadcast.

However, the large number of packets required makes the Flooding with a Hopcount Limit method unacceptable for a Multiswitch network.

One can estimate the number of packets generated by an FHL broadcast. Let h denote the maximum number of hops allowed for a broadcast message. Then, let step k, $0 < k < h$, be the action of a processor receiving a broadcast packet that has traversed k links, decrementing the hopcount field in the packet from $h - k + 1$ to $h - k$, and copying and sending the packet over its outgoing links. At step $k = 0$, the source processor sends the broadcast message with hopcount h over each of its links. At step h, a processor drops its copy of the packet.

For example, let $h = 10$. At step 1, the source processor's neighbors receive copies of the broadcast message, decrement the hopcount from 10 to 9, and forward a copy of the message over each outgoing link.

Function $r(k)$ gives an estimate for the number of packets generated by a broadcast message at step k, assuming i is the degree of the source (initial) processor, and d is the average number of outgoing links (degree) for all processors in the network:

$$r(k) = \begin{cases} i \times (d)^{k-1} & \text{for } 0 < k < h \\ 0 & \text{for } k = 0, k = h \end{cases} \qquad \text{(EQ 1)}$$

For the network in figure 3, i is 4 because the source processor has four links and sends the broadcast over all of them. The average number of outgoing links per processor in the network is 2.13.

Recursive function $s(k)$ estimates the total number of packets generated for a broadcast through step k by simply accumulating the value from previous iterations of s.

$$s(k) = \begin{cases} i \times (d)^{k-1} + s(k-1) & \text{for } 0 < k < h \\ 0 & \text{for } k = 0, k = h \end{cases} \quad \text{(EQ 2)}$$

By applying a transformation from [Knu69], $s(k)$ becomes:

$$s(k) = \begin{cases} \frac{i}{d-1}(d^k - 1) & \text{for } 0 \le k < h \\ 0 & \text{for } k = h \end{cases} \quad \text{(EQ 3)}$$

Equation 3 shows that the number of packets generated grows exponentially with respect to k, the number of hops a broadcast may take in the network.

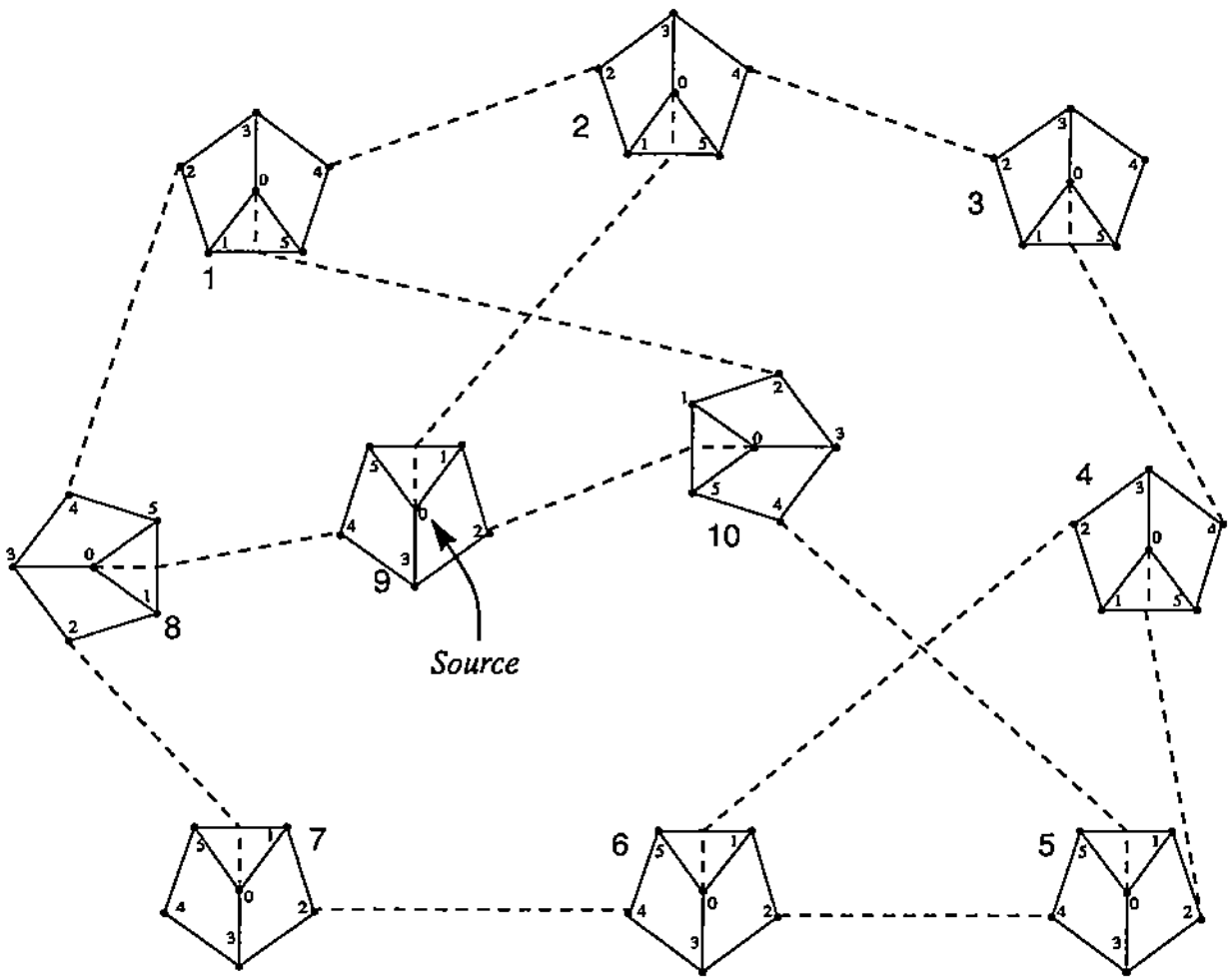

Figure 3: The Multiswitch sample network used in the examples in this paper. A solid line represents a link within a packet switch. A dashed line is a link between packet switches. A dot represents a processor. The network contains 10 packet switches, each having 6 processors.

Figure 4 contains a graph that shows the relationship between function $s(k)$ and simulation results for the sample network in figure 3. The number of packets generated in the sample network

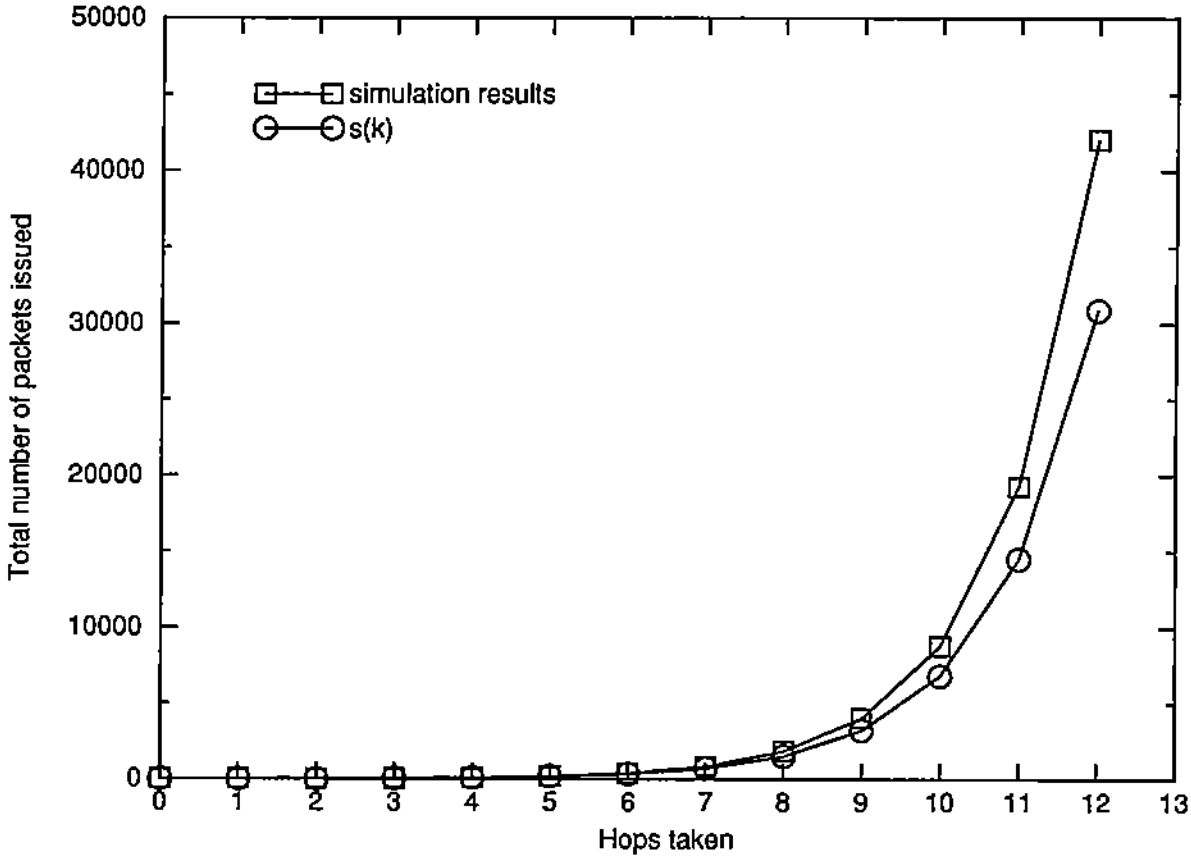

Figure 4: Number of packets issued vs. hops taken in Flooding with a Hop-count Limit algorithm.

goes above 10,000 after only eleven steps. The simulation results differ from the estimated results of $s(k)$ because each processor in the sample network does not have exactly d outgoing links.

5.3. Reverse Path Forwarding

Reverse Path Forwarding (RPF) [DM78] uses available routing information to identify and discard duplicate broadcast packets. When a processor receives a broadcast packet, the processor forwards the packet over its outgoing links if and only if the packet arrived over the link that is the last hop on the path from the source of the packet to this processor. The processor computes the last hop from the source using routing information.

When routes through a network are not symmetric, computing the RPF information may impose high memory and computational requirements upon each processor in the network. A processor normally uses routing information to compute routing tables that contain information about the next hop *toward* each processor. However, RPF requires information about the last hop on the path *from* the source to the current processor. When paths between processors are not symmetric, each processor must compute and store reverse shortest path information for each packet source.

Like SBF, RPF is susceptible to routing loops and incorrectly dropped packets. When routing

information changes while the network is delivering a broadcast, inconsistent routing tables may cause routing loops and dropped packets. Therefore, RPF should also be augmented by some other mechanism, such as hopcount limits.

RPF also does not provide a high reliability of delivery. As in SBF, a single lost packet at a processor causes an entire subtree of the source tree to miss the broadcast message. Additionally, use of RPF to deliver link-state information is dangerous. If, for any reason, RPF does not deliver the link-state information to a processor, the processor will never update its routes, so subsequent broadcasts are susceptible to routing loops and misdirected packets.

When functioning properly, RPF requires the same number of packets as Flooding With Duplicate Discard, discussed next.

6. Flooding with Duplicate Discard

The Multiswitch prototype network delivers broadcast messages using Flooding with Duplicate Discard (FDD). FDD in a Multiswitch network provides high reliability of delivery with a low number of packets per broadcast message. To identify each message, FDD requires a unique message identifier in the header of each broadcast message. FDD operates by maintaining a cache of recently-seen message identifiers, and discarding duplicate copies of each broadcast message.

FDD provides a high reliability of delivery because a single dropped packet does not prevent a subtree from receiving the broadcast message, if the subtree connects to the network by an alternate link. FDD requires that a processor only record the message id in the cache when the processor has successfully forwarded the broadcast message. (If the processor drops a packet, the message id must not be recorded.) Subsequently, the processor records the next copy of the broadcast packet (received over another link) as the first one received at the processor. A processor will miss a broadcast message only if all of the processor's neighbors drop the packets destined for the processor. Thus, in general, the probability that a processor will not receive a message is much lower than in other broadcast schemes, except Flooding with a Hopcount Limit (FHL). However, FDD does not require the large number of packets as in FHL.

In a network with n processors and e links, a broadcast message b, sent using FDD, generates

$$b(n, e) = (n-1) + 2(e-(n-1)) \qquad \text{(EQ 4)}$$

packets, when no packets are lost.

The first term in equation 4 represents packets that FDD sends over each of n - 1 links that form a source tree rooted at the source of the broadcast. These n - 1 links carry the broadcast message to all processors with minimum delay. The second term in equation 4 represents the two packets that traverse each of the other e - (n - 1) links in the network.

Intuitively, one can visualize how FDD delivers exactly one packet over each link in the source tree. As the broadcast message fans out from the source, each processor forwards its copy of the message over its links. The links that provide minimum delay carry the packets most quickly, so each processor receives its first copy of the broadcast message over the link that is the last hop in the shortest path from the broadcast source. Thereafter, each processor discards duplicate copies that arrive.

Figure 5 shows the number of packets generated by FDD and the other broadcast techniques discussed in section 5. Simulations of each technique produce the values in the graph, for the topology in figure 3. FDD and RPF each generate approximately twice as many packets as SBF, which generates the optimal number of packets. However, FDD, RPF, and SBF perform signifi-

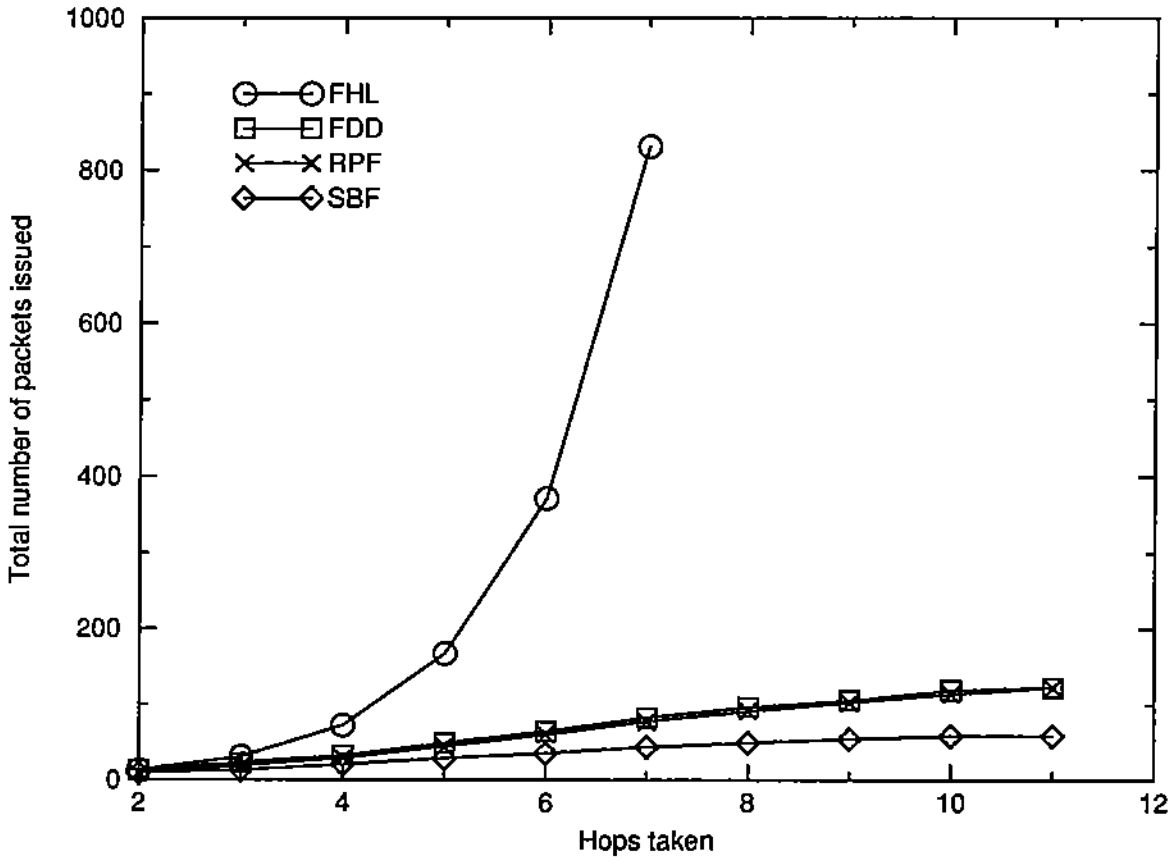

Figure 5: Comparison of number of packets generated for various broadcast schemes. Source-based forwarding (SBF) generates the optimal number of packets, 59. FDD and RPF generate 123 packets. FHL generates over 19,000 packets after 11 hops.

cantly better than FHL.

The following discussion demonstrates that exactly one packet traverses each link in the source tree from processor s and that exactly two packets traverse the remaining links in the network. Because a source tree always contains n - 1 links, the discussion demonstrates that equation 4 is correct.

The discussion first demonstrates that exactly one message traverses each of the n - 1 links that form the source tree of s. Then, we argue that exactly two messages traverse each link that is not part of the source tree of s. We argue that the links not in the source tree of s cannot carry zero, one, or more than two packets, and thus they must carry exactly two packets.

6.1. Arguments for Correctness of Equation 4

Given a network containing n processors and e communication links, make the following simplifying assumptions about the operation of the network:

(1) Packet processing time at a processor is negligible.

(2) Processors do not drop packets.

(3) All communication links are bidirectional.

(4) If two packets arrive at a processor at the same time, the processor arbitrarily orders the packets and processes them in sequence.

(5) There is at most one bidirectional link between any pair of processors, m and n, and it is labeled mn or nm.

(6) The network connections are fixed during the broadcast.

Consider a link mn in the network from processor m to processor n. Assume that link mn is the link with minimum delay from processor m, so that link mn is in the source tree of s. Because processor m is in the source tree of s and link mn is the minimum link from processor m, by the principle of optimality [Raj92] and assumption (1), the packet from processor m to processor n traversing link mn must either **(a)** be the first packet to reach processor n, or **(b)** reach processor n at exactly the same time as another packet traversing an alternate link to processor n. In either case, only one packet traverses link mn.

In case **(a)**, the packet traversing mn reaches processor n before any other packet in the broadcast. Because the packet traversing mn reaches processor n first, processor n caches the broadcast identifier and forwards copies of the packet over all links except link mn. Then, processor n will discard all subsequent copies of the broadcast message it receives. Thus, processor n will never send a message over link nm. Because processor m has also received the broadcast message and cached the message identifier, processor m will also never again forward the broadcast message over link mn. Thus, one and only one packet traverses link mn.

In case **(b)**, processor n receives two copies of the broadcast message from two distinct links at exactly the same time. By assumption (4), and without loss of generality, assume that processor n selects the packet from link mn first and thus processor n becomes part of the source tree via link mn. Then, case **(b)** reduces to case **(a)**. Thus, one and only one packet traverses link mn.

Now, consider a link mn in the network that is *not* in the source tree of s. First, there cannot be zero packets sent over link mn because when no packets are droped, all processors receive a packet at least once and each processor responds by forwarding the packet over all links except the link on which the packet arrived. Thus, all links carry the broadcast message at least once.

Next, the link must carry at least one packet. Assume, without loss of generality, that the first packet, p, traverses link mn from processor m to processor n. If link mn is not in the source tree of s, then processor n must be reachable from s via an alternate, shorter path that does not include link mn. But, if packet p is the first to traverse link mn in either direction, then when p arrives at processor n, processor n must be busy processing another copy of the packet, q, that arrived earlier via another link. When processor n finishes processing packet q, processor n will send copies over all other links, including link nm. Processor n will then accept and process packet p and discard it. Thus, two packets traverse link mn and thus, in general, any link not in the source tree of s.

Observe that the two packets must traverse the link in opposite directions. Additionally, note that the two packets, p and q, traverse the link mn *simultaneously*, or, at least, one of the packets, say p, traverses the link mn and is not processed before packet q traverses the link in the opposite direction.

Finally, there will never be more than two packets sent over any link. A link has only two endpoints and each of these endpoints forwards a broadcast only once. Thus, only two packets can be sent over each link for a single broadcast.

To summarize, the number of packets sent during a Flooding With Duplicate Discard broadcast from a processor s is one packet for each link in the source tree of s, plus two packets for each link not in the source tree of s. Thus, equation 4 accurately estimates the total number of packets

sent in an FDD broadcast.

6.2. Observations

Notice, from equation 4, that the number of packets required to propagate a broadcast using FDD is independent of the source of the broadcast.

Also notice that the number of packets required for an FDD broadcast is independent of network configuration, given that the number of processors and the number of links each remains constant. Thus, the efficiency of the operation of FDD is independent of network diameter and the degree of the source processor.

Additionally, an FDD broadcast through a network dynamically outlines a source tree. The source tree consists of links traversed by only one packet during the broadcast. Using this observation, a simple extension to FDD allows a network to implement source-based forwarding without having to use Dijkstra's algorithm to compute outgoing link subsets. The appendix of this paper contains the extension.

6.3. Cache Size and Cache Entry Lifetime

Dalal and Metcalfe [DM78] dismiss the use of FDD because they claim that determining the size of the cache and the time each cache entry should remain valid is too difficult. If the cache size is too small, a processor will not be able to identify a duplicate copy of a broadcast, and FDD will generate a large number of unnecessary copies of the broadcast, as in FHL. If the cache size is too large, cache memory is wasted.

Observations from the discussion in section 6.1 provide hints for determining a cache size. Each link carries at most two copies of each broadcast message. Also, if a broadcast message traverses a link twice, the packets travel in opposite directions, and, in essence, the packets travel over the link simultaneously. (It may be that a packet from processor m to processor n is waiting to be processed at processor n while processor n forwards a duplicate packet back over link nm to processor m).

Thus, in a system in which no processors are misbehaving (by holding packets for an unusally long period of time), a duplicate packet may only arrive at a processor n within the maximum round-trip time between processor n and its neighbors. Thus, a processor only needs to cache a broadcast identifier for the maximum round-trip time after receiving the original broadcast message. Figure 6 illustrates the situation in which a duplicate arrives at approximately one round-trip time between processor n and m.

RTT, the round-trip time, is the maximum time required for a processor to send a packet to any one of its neighbors and receive a copy back. The cache on a processor needs to store broadcast identifiers only for the RTT. After the RTT, a processor may discard the cache entry because the processor cannot receive another copy of the broadcast message. Estimating RTT may be difficult because of variable queueing delays at the processors.

Much work has been done on monitoring and determining round-trip time estimates [KP87, Mor79, Pos81, VHS84]. Round-trip time estimates vary greatly in packet switching networks due to sudden bursts of traffic arriving at a processor. Therefore, to simplify the FDD algorithm, use a fixed maximum packet lifetime value as the RTT estimate. A packet switch that enqueues a packet for longer than the maximum packet lifetime must discard the packet instead of sending it. Then, each cache entry need only remain valid for the fixed RTT value.

Additionally, to limit the size of the cache, enforce an interpacket gap of RTT (or maximum

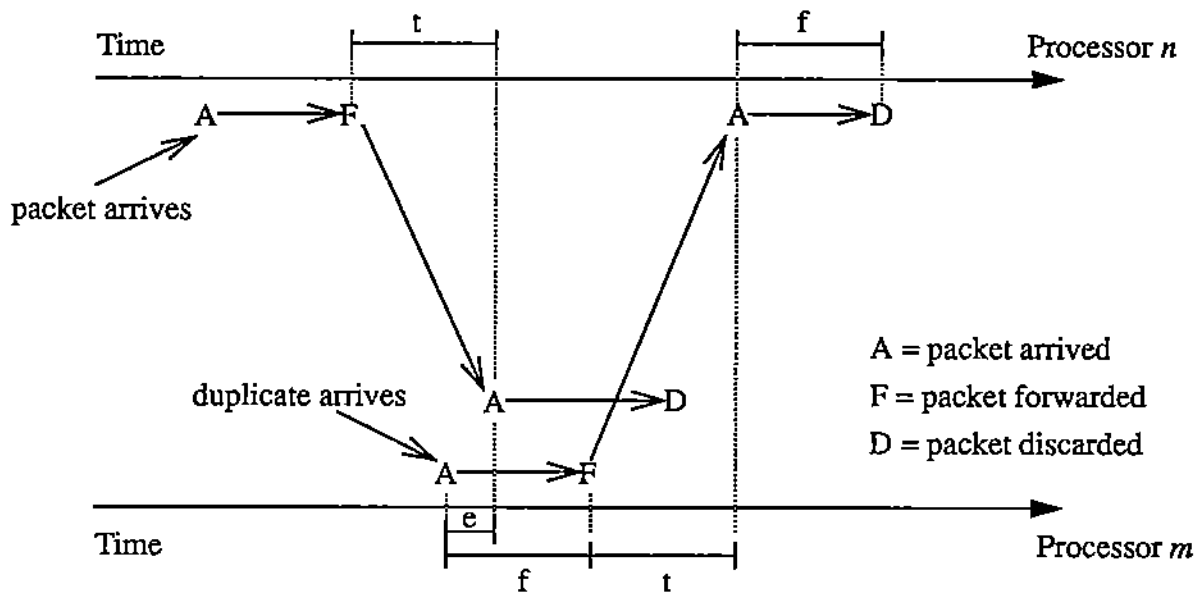

Figure 6: Illustration of duplicate arriving after slightly less than one round-trip time delay. The original packet arrives first at processor *n* and then later at processor *m*. Processor *n* forwards the packet to processor *m* which is already processing its own copy of the broadcast. The duplicate copy traverses link *mn* and is processed by processor *n* at time 2(t + f) - e. Note that one exact round-trip time is 2(t + f).

packet lifetime) between successive broadcasts from each processor. With an interpacket gap time between broadcasts, a processor need only cache one broadcast identifier from any one source at any given time. Then, the maximum cache size is a linear function with respect to the number of processors in the network.

7. Summary

The Flooding With Duplicate Discard (FDD) technique for broadcasting messages in a point-to-point, store-and-forward network provides both high reliability of delivery and high efficiency. When compared to other techniques in common use, FDD is the best known algorithm for implementation in a Multiswitch network. A FDD broadcast generates a fixed number of packets, regardless of the source of the broadcast. A FDD broadcast outlines a source tree rooted at the source of the broadcast. Two simplifying assumptions allow efficient implementation of FDD by limiting cache size and cache entry lifetimes. A new, more efficient implementation of source-based forwarding uses knowledge gained from the behavior of FDD in a point-to-point network.

8. References

[Dij59] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269-271, 1959.

[DM78] Yogen K. Dalal and Robert M. Metcalfe. Reverse Path Forwarding of Broadcast Packets. *Communications of the ACM*, 21(12):1040-1048, December 1978.

[Knu69] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Publishing Compnay, 1969.

[KP87] Phil Karn and Craig Partridge. Imporving Round-Trip Time Estimates in Reliable Transport Protocols. In *Proceedings of ACM SIGCOMM '87*, pages 2-7, August 1987.

[Mor79] Robert J.T. Morris. Fixing timeout intervals for lost packet detection in computer communication networks. In *AFIPS Conference Proceedings*. National Computer Conference, 1979.

[Moy91] J. Moy. OSPF Version 2, July 1991. RFC 1247.

[MRR80] J.M. McQuillan, I. Richer, and E.C. Rosen. The New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711-719, May 1980.

[Nar88] T. Narten. *Best Effort Delivery in Connectionless Networks*. PhD thesis, Department of Computer Science, Purdue University, West Lafayette, IN, August 1988.

[Pos81] J. Postel. Transmission Control Protocol -- DARPA Internet program protocol specification, September 1981. RFC 793.

[Raj92] Bala Rajagopalan. Reliability and Scaling Issues in Multicast Communication. In *Proceedings of ACM SIGCOMM '92*, pages 188-198, October 1992.

[VHS84] David Velten, Robert Hinden, and Jack Sax. Reliable Data Protocol, July 1984. RFC 908.

9. Appendix: Source-Based Forwarding using FDD

FDD can be extended to efficiently build source-based forwarding tables in a network where the chosen link metric is link delay. Recall that source-based forwarding requires each processor to maintain a set of outgoing links for each source processor in the network. A processor must compute an outgoing link set, usually using Dijkstra's algorithm and the full topology information available at each processor. However, the computation can be avoided by using FDD to find the source tree.

A processor builds and modifies an outgoing link set by noticing when one of its links has carried a duplicate packet. Recall that each link that carries two packets does not belong in the source tree. If processor m receives a broadcast message from source s, processor m sends the broadcast to processor n and initializes the outgoing link set for source s to include all links. If processor m receives another copy of the broadcast message from a neighbor processor, say processor n, link mn does not belong in the source tree from s and the link is removed from the outgoing link set at both processors m and n.

When the topology of the network changes, the source trees for the network may also change. The outgoing link sets stored at each processor may be incorrect. After a topology change, the source of a broadcast should indicate within the next broadcast packet that the processors' outgoing link sets may be incorrect and new ones should be built.